A2 cont.

In the brief description of FIG. 7 beginning on page 3, line 3, please amend as follows:

FIG. 7 is a flowchart showing the operations involved in a remote write lock release, according to an embodiment of the present invention.

IN THE SPECIFICATION:

Please delete the paragraph beginning with "Figure 3 shows the steps involved in making" on page 7, line 5, and replace with the following:

A3

Figure 3 shows the operations involved in making a local write lock **210** request. If a write request is already pending, as questioned in operation **300**, the system must wait until its pending status drops, as depicted in operation **305**, and then increment the pending semaphore and continue, as shown in operation **310**. That is, the semaphore operates as a variable with a value that indicates the status of common resource. It locks the resource that is being used. The process needing the resource checks the semaphore to determine the resource's status and then decides how to proceed. As described in operation **315**, if the local module already has the write lock **210**, then the system increments a local write lock **210** count. If the write lock **210** is set by another object management system, the system must wait, as illustrated in operation **320**, for it to be released. If an object management system has just released its local lock and is in the process of sending the release to remote object management systems, as shown in operation **325**, the system must wait until the release option is complete. A clear-pending lock is employed here. Next, as depicted in operation **330**, the lock contention mutex is locked and a random number is created for resolving write lock **210** contention. As shown in operation **335**, the system then checks a variable dedicated to write lock **210** arbitration counting. If the value is greater than zero, a request has been received by a remote object management between operations **320** and

325. As such, the lock contention mutex is released and the flow returns to operation **320**. Otherwise, the arbitration count is incremented, as illustrated in operation **350**. As described in operation **355**, the lock's **210** arbitration identification is set to the local object management system identification. Its priority is also set through the generation of a bound random number. The lock contention mutex is then released, as shown in operation **360**. For each connection returned through an iterator provided by the list controller **130**, a write lock **210** request is made

A3 cont.

to the remote reader and writer's lock, as illustrated in operation **365**. Operation **370** then examines whether the request failed. If a request fails because of contention in a remote module, the local write lock count is decremented, as shown in operation **375**, and the process repeats from operation **320,** as shown in operation **380.** Instead of creating a new random number for resolving write lock **210** contention, the request' s priority is bumped beyond the upper boundary of the random number range. This guarantees that requests re-entering arbitration are afforded higher priority than new requests. The size of the random number generation range must be low enough to allow the arbitration to bump several times without integer overflow; enough to accommodate the maximum size of the pending request queue. If the request did not fail, as depicted in operation **385**, when all remote modules have provided the write lock **210** to the requesting module, the process returns without setting the write lock **210** in the lock component. This allows the local module to grant read locks **200**.

Please delete the paragraph beginning with "Figure 4 shows the steps involved in making a local" on page 8, line 15, and replace with the following:

Figure 4 shows the operations involved in making a local read lock **200** request. This function does not result in any request being passed over the interface definition language interface **110**. The algorithm is as follows. As shown in operation **410**, the request is sent to the local lock component. Operation **420** then examines whether a remote module owns the write lock **210**. If a remote module does not own the write lock **210**, the lock is granted, as shown in operation **430**. If a remote module owns the write lock **210**, as depicted in operation **440**, the write lock **210** is blocked. Operation **450** shows that the request is then retried. If the available time for the request expires, as illustrated in operation **460**, the process repeats until the maximum number of retries is reached.

Please delete the paragraph beginning with "Figure 5 shows the steps involved in making a remote" on page 8, line 23, and replace with the following:

A3 cont.

Figure 5 shows the operations involved in making a remote write lock **210** request. This request is received over the interface definition language interface **110**. Each lock module is responsible for ensuring that it never sends a request to remote object management systems if it already owns the write lock **210**. Therefore, more than one call to this function without an intervening release of the write lock **210** will only occur during write lock **210** contention. This allows the ownership of the write lock **210** to be changed during the write contention interval, where the write contention interval is, the time it takes for the requesting module to be granted the write lock **210** by all other modules.

Please delete the paragraph beginning with "As shown in step **510**, the lock contention" on page 9, line 8, and replace with the following:

As shown in operation **510**, the lock contention mutex is locked, and the arbitration count is checked. Operation **520** examines whether this value is greater than zero. If the value is not

A3 cnd.

greater than zero, the count is incremented, as illustrated in operation **530**. If the value is greater than zero, a request has been previously received. The contention for the write lock **210** must thus be resolved, as shown in operation **540**. The resolve write lock contention function is called when a remote object management system requests a write lock **210**, yet the write lock **210** contention variables, priority and identification, have either been set by a local request or by a previous invocation of the write lock **210** arbitration function.

A3 cont.

Please delete the paragraph beginning with "Step **550** examines whether the remote object management system" on page 10, line 5, and replace with the following:

A4

Operation **550** examines whether the remote object management system prevailed in arbitration. As shown in operation **560**, if the remote object management system loses arbitration, the lock contention mutex is released and information about the arbitration winner to the requesting object management system is returned. Otherwise, as illustrated in operation **570**, the request is sent to the lock component, which will block until all outstanding read locks **200** are released. The lock contention mutex is then released, as depicted in operation **580**. The operations outlined above cannot be executed in one function because remote requests are received in the interface definition language skeleton code, which must not block for an indeterminate period. Therefore, the incoming request must be asynchronous to the reply.

Please delete the paragraph beginning with "Figure 6 shows the steps involved in a local write lock" on page 10, line 14, and replace with the following:

A4 cont.

Figure 6 shows the operations involved in a local write lock **210** release. As shown in operation **610**, a clear pending lock is set to prevent the local object management system from processing another request until the current lock has been released from all remote object management systems. The write lock **210** contention mutex is then locked and the write lock **210** count is decremented, as illustrated in operation **620**. Operation **630** then examines whether the write lock **210** count is zero. If the write lock **210** count is zero, as shown in operation **640**, the write lock **210** contention variables, identification and priority, must be cleared. Otherwise, some other module on the local object management system has a write lock **210**. In that case, the lock contention mutex must be released, as depicted in operation **650**. Operation **660** describes the calling of the write lock **210** release functions on remote modules over the interface definition language interface **110**. This is achieved in two stages—one to clear the arbitration variables in the remote object management systems and one to enable arbitration for the next request. Otherwise, in a system of three or more object management systems, a remote object management system could arbitrate against stale arbitration values. The clear pending lock is then released, as shown in operation **670**.

A4 cont.

Please delete the paragraph beginning with "Figure 7 shows the steps involved in a remote write lock" on page 11, line 20 and replace with the following:

Figure 7 shows the operations involved in a remote write lock **210** release. As explained in the preceding section, this activity occurs in two stages. First, as described in operation **710**, the clear pending lock is set to prevent the local object management system from processing another request until the current lock has been released from all remote object management systems. The write lock **210** contention mutex is then locked and the write lock **210** contentions variables, write identification and priority, are cleared, as shown in operation **720**. The writer's lock **210** is then cleared from the local lock component, as illustrated in operation **730**. The lock

A4 cont'd

contention mutex is released, as described in operation **740**. In the second stage, the clear pending lock is cleared, as shown in operation **750**. The second stage occurs after the owner of the write lock **210** on all connected object management systems has performed the first stage.

Please delete the paragraph beginning with "While the above description refers" on page 12, line 7 and replace with the following:

A5

While the above description refers to particular embodiments of the present invention, it will be understood to those of ordinary skill in the art that modifications may be made without departing from the spirit thereof. The accompanying claims are intended to cover any such modifications as would fall within the true scope and spirit of the embodiments of the present invention.

Please delete the paragraph beginning with "The presently disclosed embodiments" on page 12, line 11 and replace with the following:

The presently disclosed embodiments are therefore to be considered in all respects as illustrative and not restrictive; the scope of the embodiments of the invention being indicated by the appended claims, rather than the foregoing description. All changes that come within the meaning and range of equivalency of the claims are therefore intended to be embraced therein.

IN THE CLAIMS:

Please amend claims 27 through 36 as follows:

27. (Amended) An article comprising:

A6